**12.3**

#include <iostream>

#include <string>

using namespace std;

// creates a sort function of a generic type

template <typename T>

void selectionSort(T list[], int size)

{

for (int i = 0; i < size - 1; i++)

{

// Find the minimum in the list

T currentMin = list[i];

int currentMinIndex = i;

for (int j = i + 1; j < size; j++)

{

if (currentMin > list[j])

{

currentMin = list[j];

currentMinIndex = j;

}

}

// Swap list[i] with list[currentMinIndex] if necessary;

if (currentMinIndex != i)

{

list[currentMinIndex] = list[i];

list[i] = currentMin;

}

}

}

// lets the array do a binary search of a generic type

template <typename T>

int binarySearch(const T list[], T key, int listSize)

{

int low = 0;

int high = listSize - 1;

while (high >= low)

{

int mid = (low + high) / 2;

if (key < list[mid])

high = mid - 1;

else if (key == list[mid])

return mid;

else

low = mid + 1;

}

return -1;

}

// prints the array of generic type

template <typename T>

void printArray(T list[], int size)

{

for (int i = 0; i < size; i++)

cout << list[i] << " ";

cout << endl;

}

int main()

{

// tests the functions with int, double, and string arrays

int a[] = { 10,7,3,15 };

double b[] = { 5.8,6.4,2.3,9.7 };

string c[] = { "mickey", "goofy", "pluto", "donald" };

// sorts and prints each array, then shows the results of the binary search

selectionSort(a, 4);

printArray(a, 4);

cout << "7 is at index " << binarySearch(a, 7, 4) << endl;

cout << "5 is at index " << binarySearch(a, 5, 4) << endl;

cout << endl;

selectionSort(b, 4);

printArray(b, 4);

cout << "2.3 is at index " << binarySearch(b, 2.3, 4) << endl;

cout << "4.5 is at index " << binarySearch(b, 4.5, 4) << endl;

cout << endl;

selectionSort(c, 4);

printArray(c, 4);

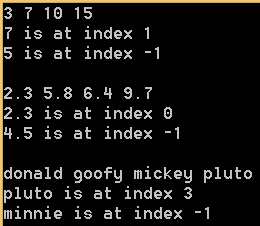
cout << "pluto is at index " << binarySearch(c, string("pluto"), 4) << endl;

cout << "minnie is at index " << binarySearch(c, string("minnie"), 4) << endl;

system("pause");

return 0;

}



**12.25**

**Date Header**

#ifndef DATE\_H

#define DATE\_H

class Date

{

public:

// month = 1 - 12, day = 1 - 31

Date();

Date(int year, int month, int day);

// get functions for the year, month, and day

int getYear() const;

int getMonth() const;

int getDay() const;

// print function for date

void printDate() const;

private:

int year;

int month;

int day;

};

#endif

**Date Implementation**

#include "date.h"

#include <iostream>

#include <ctime>

using namespace std;

Date::Date()

{

year = 2017;

month = 12;

day = 16;

}

Date::Date(int year, int month, int day)

{

this->year = year;

this->month = month;

this->day = day;

}

int Date::getYear() const

{

return year;

}

int Date::getMonth() const

{

return month;

}

int Date::getDay() const

{

return day;

}

void Date::printDate() const

{

cout << getMonth() << "-" << getDay() << "-" << getYear();

}

**Transaction Header**

#ifndef TRANSACTION\_H

#define TRANSACTION\_H

#include "date.h"

#include <string>

using namespace std;

class Transaction

{

public:

Transaction(char t, double a, double b, string des);

void setDate(Date d);

Date getDate() const;

void setType(char t);

char getType() const;

void setAmount(double a);

double getAmount() const;

void setBalance(double b);

double getBalance() const;

void setDescription(string des);

string getDescription() const;

void printTransaction() const;

private:

Date date;

char type;

double amount;

double balance;

string description;

};

#endif

**Transaction Implementation**

#include "transaction.h"

#include <string>

#include <iostream>

#include <iomanip>

using namespace std;

Transaction::Transaction(char t, double a, double b, string des)

{

type = t;

amount = a;

balance = b;

description = des;

printTransaction(); // allows the created transaction to print after execution

}

void Transaction::setDate(Date d)

{

date = d;

}

Date Transaction::getDate() const

{

return date;

}

void Transaction::setType(char t)

{

type = t;

}

char Transaction::getType() const

{

return type;

}

void Transaction::setAmount(double a)

{

amount = a;

}

double Transaction::getAmount() const

{

return amount;

}

void Transaction::setBalance(double b)

{

balance = b;

}

double Transaction::getBalance() const

{

return balance;

}

void Transaction::setDescription(string des)

{

description = des;

}

string Transaction::getDescription() const

{

return description;

}

void Transaction::printTransaction() const

{

cout << left << setw(9) << description

<< setw(4) << amount << "Current balance is $" << balance << endl;

}

**Account Header**

#ifndef ACCOUNT\_H

#define ACCOUNT\_H

#include <string>

#include <vector>

#include "transaction.h"

using namespace std;

class Account

{

public:

Account();

Account(int newId, double newBalance, double newRate);

Account(string newName, int newId, double newBalance);

int getId() const;

void setId(int newId);

double getBalance() const;

void setBalance(double newBalance);

double getRate() const;

void setRate(double newRate);

double getMonthlyInterestRate() const;

void setName(string name);

string getName() const;

void withdraw(double amount); // will automatically show the transaction after execution

void deposit(double amount); // will automatically show the transaction after execution

void printSummary() const; // prints the account summary - name, interest rate, balance

private:

int id;

double balance;

static double annualInterestRate;

string name;

vector<Transaction> transactions;

};

#endif

**Account Implementation**

#include "account.h"

#include <iostream>

#include <iomanip>

using namespace std;

Account::Account()

{

id = 0;

balance = 0;

annualInterestRate = 0;

}

Account::Account(int newId, double newBalance, double newRate)

{

id = newId;

balance = newBalance;

annualInterestRate = newRate;

}

Account::Account(string newName, int newId, double newBalance)

{

name = newName;

id = newId;

balance = newBalance;

}

int Account::getId() const

{

return id;

}

void Account::setId(int newId)

{

id = newId;

}

double Account::getBalance() const

{

return balance;

}

void Account::setBalance(double newBalance)

{

balance = newBalance;

}

double Account::getRate() const

{

return annualInterestRate;

}

void Account::setRate(double newRate)

{

annualInterestRate = newRate;

}

double Account::getMonthlyInterestRate() const

{

return annualInterestRate / 12;

}

void Account::setName(string name)

{

this->name = name;

}

string Account::getName() const

{

return name;

}

void Account::withdraw(double amount)

{

if (balance > amount && amount > 0)

{

balance -= amount;

transactions.push\_back(Transaction('W', amount, balance, "Withdraw"));

}

else if (amount < 0)

cout << "Please try again with a positive number.\n";

else

cout << "Insufficient balance. Please try again.\n";

}

void Account::deposit(double amount)

{

if (amount > 0)

{

balance += amount;

transactions.push\_back(Transaction('D', amount, balance, "Deposit"));

}

else

cout << "Please try again with a positive number.\n";

}

void Account::printSummary() const

{

cout << "Account Summary:\n" << left << setw(22) << "Name: " << getName()

<< "\nAnnual interest rate: " << getRate() \* 100

<< setw(25) << " %\nBalance:" << "$" << getBalance() << endl;

}

double Account::annualInterestRate = 0.015;

**Main**

#include "date.h"

#include "transaction.h"

#include "account.h"

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

// creates an Account object

Account a("George", 1122, 1000);

a.setRate(0.015);

// prints the account summary

a.printSummary();

cout << "\nThe following are the transactions on ";

// creates and prints the date for the transaction

Date d(2017, 12, 18);

d.printDate();

cout << ":" << endl;

// executing the deposit and withdraw functions

a.deposit(30);

a.deposit(40);

a.deposit(50);

a.withdraw(5);

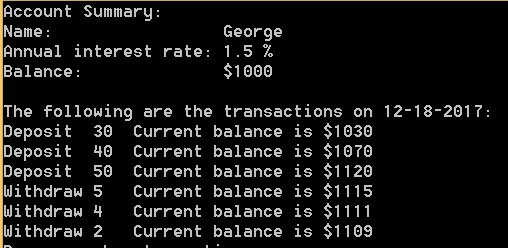
a.withdraw(4);

a.withdraw(2);

system("pause");

return 0;

}

****

**12.32**

#include <iostream>

#include <vector>

using namespace std;

// tells the vector to do a linear search

template <typename T>

int lSearch(vector<T> v, T key)

{

for (int i = 0; i < v.size(); i++)

if (key == v[i])

return i;

return -1; // search failed

}

// removes the duplicates that might exist in a vector

template <typename T>

void removeDuplicate(vector<T>& v)

{

vector<T> temp;

for (int i = 0; i < v.size(); i++)

{

int index = lSearch(temp, v[i]);

if (index == -1)

temp.push\_back(v[i]);

}

v = temp;

}

// prints the vector

template<typename T>

void printVector(vector<T> v)

{

for (int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

}

int main()

{

// prompts the user to enter 10 integers

cout << "Enter ten integers: ";

vector<int> intVector;

int a[10];

for (int i = 0; i < 10; i++)

{

cin >> a[i];

intVector.push\_back(a[i]);

}

// calls the removeDuplicate and printVector functions

removeDuplicate(intVector);

printVector(intVector);

system("pause");

return 0;

}



**14.3**

**Header**

#ifndef CIRCLE\_H

#define CIRCLE\_H

class Circle

{

public:

Circle();

Circle(double);

double getArea() const;

double getRadius() const;

void setRadius(double);

static int getNUmberOfObjects();

void printCircle() const;

// overload the operation <, <=, ==, !=, >, >=

bool operator<(const Circle& right);

bool operator<=(const Circle& right);

bool operator==(const Circle& right);

bool operator!=(const Circle& right);

bool operator>(const Circle& right);

bool operator>=(const Circle& right);

private:

double radius;

static int numberOfObjects;

};

#endif

**Implementation**

#include "CircleWithConstantMemberFunctions.h"

#include <iostream>

using namespace std;

int Circle::numberOfObjects = 0;

// Construct a circle object

Circle::Circle()

{

radius = 1;

numberOfObjects++;

}

// Construct a circle object

Circle::Circle(double newRadius)

{

radius = newRadius;

numberOfObjects++;

}

// Return the area of this circle

double Circle::getArea() const

{

return radius \* radius \* 3.14159;

}

// Return the radius of this circle

double Circle::getRadius() const

{

return radius;

}

// Set a new radius

void Circle::setRadius(double newRadius)

{

radius = (newRadius >= 0) ? newRadius : 0;

}

// Return the number of circle objects

int Circle::getNUmberOfObjects()

{

return numberOfObjects;

}

void Circle::printCircle() const

{

cout << "Circle has a radius of " << radius << endl;

}

bool Circle::operator<(const Circle& right)

{

if (this->radius < right.radius)

return true;

else

return false;

}

bool Circle::operator<=(const Circle& right)

{

if (this->radius <= right.radius)

return true;

else

return false;

}

bool Circle::operator==(const Circle& right)

{

if (this->radius == right.radius)

return true;

else

return false;

}

bool Circle::operator!=(const Circle& right)

{

if (this->radius != right.radius)

return true;

else

return false;

}

bool Circle::operator>(const Circle& right)

{

if (this->radius > right.radius)

return true;

else

return false;

}

bool Circle::operator>=(const Circle& right)

{

if (this->radius >= right.radius)

return true;

else

return false;

}

**Main**

#include "CircleWithConstantMemberFunctions.h"

#include <iostream>

using namespace std;

int main()

{

// creates two Circle objects

Circle c1(5);

Circle c2(7.5);

cout << "c1's radius is " << c1.getRadius() << endl;

cout << "c2's radius is " << c2.getRadius() << endl;

cout << endl;

// compare c1 and c2

cout << "Is c1's radius < c2's radius? " << boolalpha << (c1 < c2) << endl;

cout << "Is c1's radius <= c2's radius? " << boolalpha << (c1 <= c2) << endl;

cout << "Is c1's radius == c2's radius? " << boolalpha << (c1 == c2) << endl;

cout << "Is c1's radius != c2's radius? " << boolalpha << (c1 != c2) << endl;

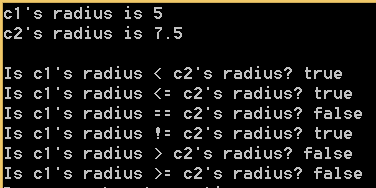
cout << "Is c1's radius > c2's radius? " << boolalpha << (c1 > c2) << endl;

cout << "Is c1's radius >= c2's radius? " << boolalpha << (c1 >= c2) << endl;

system("pause");

return 0;

}

****